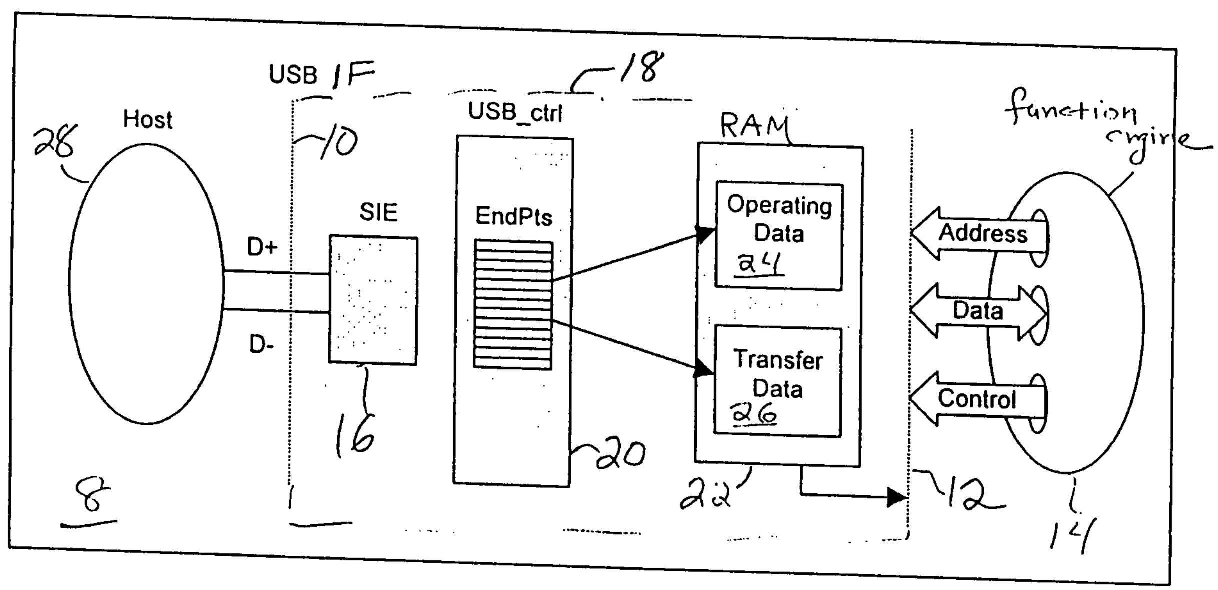

Host
28
D+
D-
USB IF
10
SIE
16
USB_ctrl
18
EndPts
20
RAM
Operating Data
24
Transfer Data
26
22
12
Address
Data
Control
function engine
14
8

FIG. 1

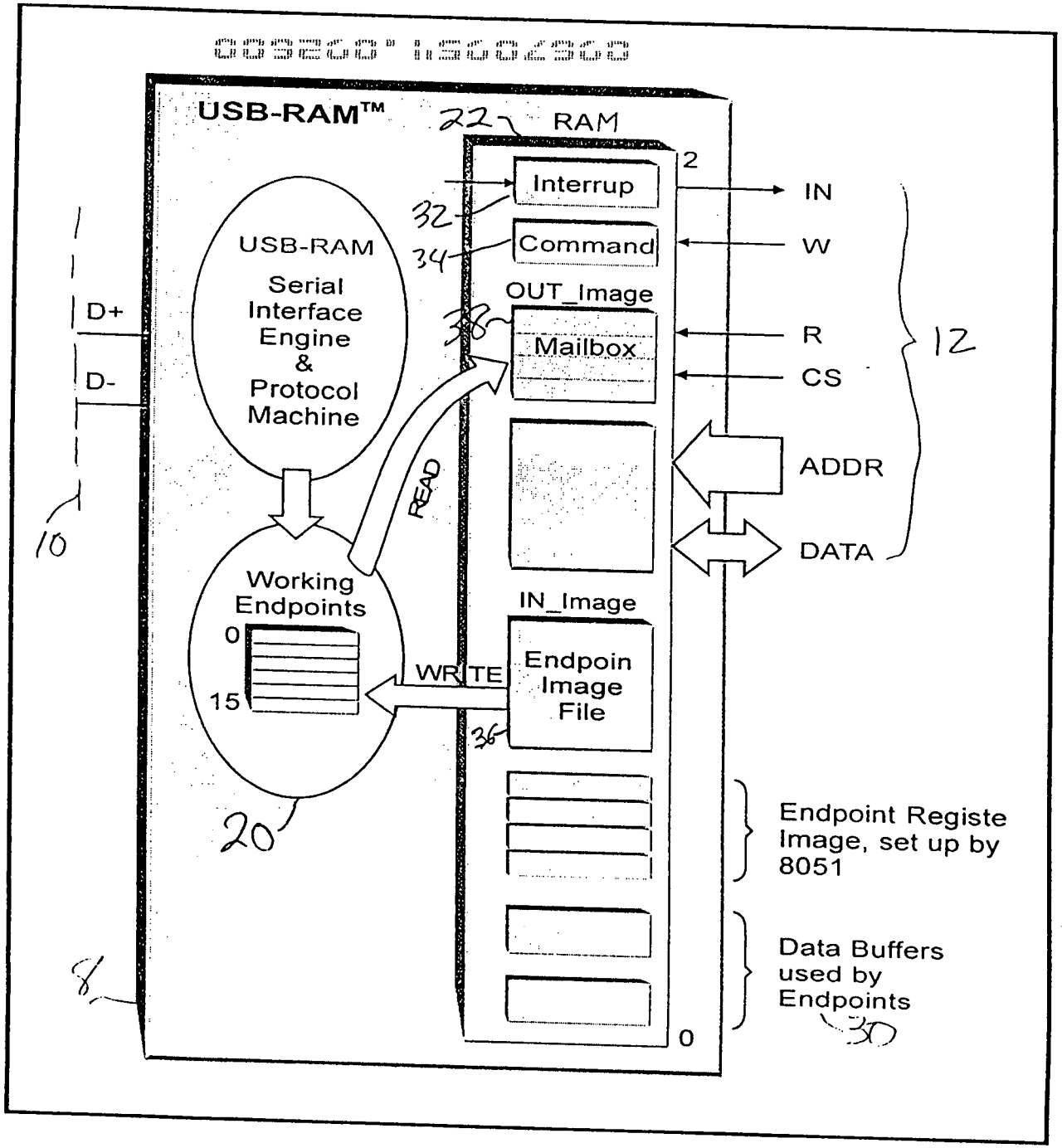
USB-RAM™
22
RAM
2
Interrup
32
Command
34
OUT_Image
38
Mailbox
IN
W
R
CS
12
ADDR
DATA
USB-RAM
Serial
Interface
Engine
&
Protocol
Machine
D+
D-
10
READ
Working
Endpoints
0
15
20
IN_Image
Endpoin
Image
File
WRITE
36
Endpoint Registe
Image, set up by
8051
Data Buffers
used by
Endpoints
30
8
0

FIG. 2

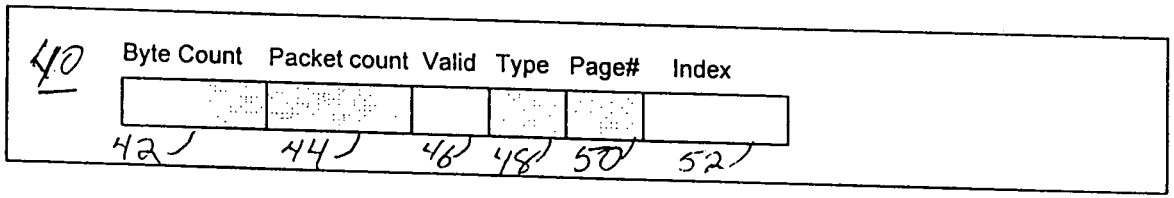
40
Byte Count
Packet count
Valid
Type
Page#
Index
42
44
46
48
50
52

FIG. 3

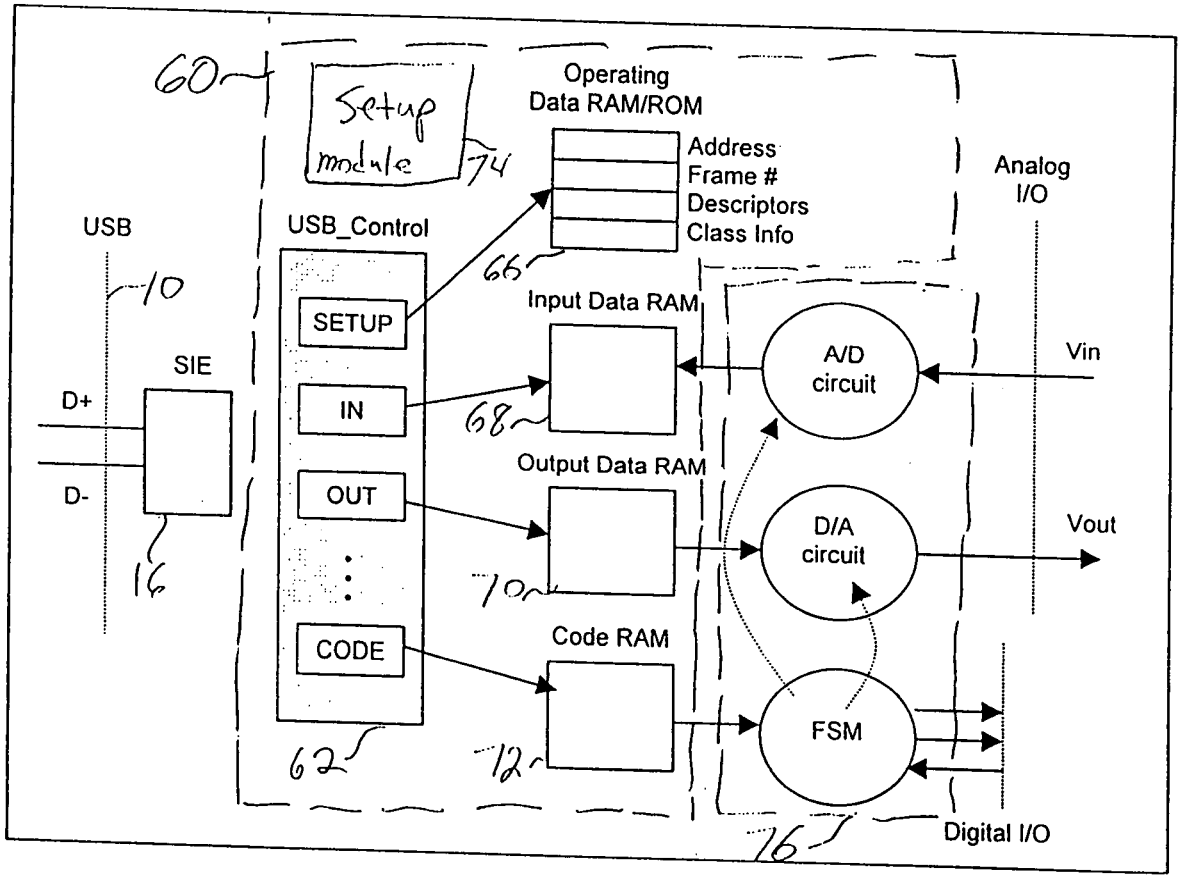
60
Setup module
74
Operating Data RAM/ROM
Address
Frame #
Descriptors
Class Info
66
Analog I/O
USB
10
USB_Control
SETUP
IN
OUT
CODE
62
Input Data RAM
68
Output Data RAM
70
Code RAM
72
SIE
D+
D-
16
A/D circuit
Vin
D/A circuit
Vout
FSM
Digital I/O
76

FIG. 4

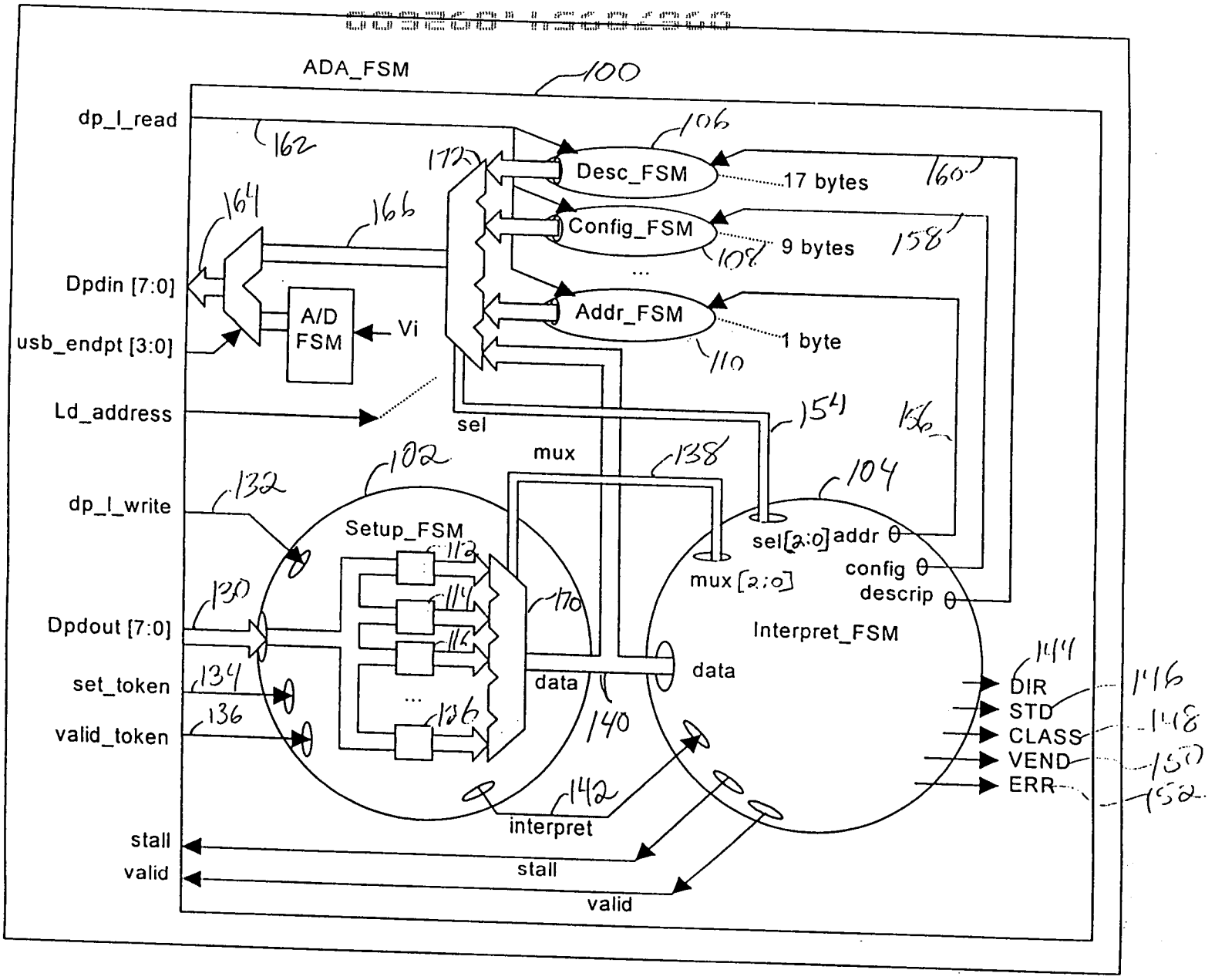
ADA_FSM
100
dp_l_read
162
106
Desc_FSM
17 bytes
160
172
164
166
Config_FSM
9 bytes
108
158
Dpdin [7:0]
A/D FSM
Vi
Addr_FSM
1 byte
usb_endpt [3:0]
110
Ld_address
154
156
sel
mux
102
138
104
dp_l_write
132
Setup_FSM
sel[2:0]
addr
config
descrip
mux [2:0]
112
130
114
170
Dpdout [7:0]
116
Interpret_FSM
data
data
144
set_token
134
DIR
146
STD
148
CLASS
valid_token
136
126
140
VEND
150
ERR
152
142
interpret
stall
valid
stall
valid

FIG. 5

```
module ADA (clk,reset,dpdi,dpdout,usb_endpt,dp_l_read,
          dp_l_write,set_token,valid_token,stall,Vo,Vi)
output [7:0] dpdi;
output stall;
input [7:0] dpdout;
input [3:0] usb_endpt;
input set_token, valid_token, dp_l_read, dp_l_write;

SETUP_FSM

setup_FSM (.dpdout(dpdout),.clk (clk),.reset(reset),
.data (data),.mux (mux).dp_l_write(dp_l_write), .set_token
(set_token),.valid_token (valid_token));

INTERPRET_FSM

interpret_FSM(.clk (clk),.reset (reset), .data (data), .stall
(stall), .mux (mux),.sel (sel), .address (address), .config
(config), .descrip (descrip), .DIR (DIR), .STD (STD), .etc. );

DESCRIP_FSM

descrip_FSM (.d_data (d_data),.dp_l_read (dp_l_read), .descrip
(descrip),.reset (reset),.clk  (clk));

CONFIG_FSM

config_FSM(.c_data (c_data),.dp_l_read (dp_l_read),
.config (config),.reset (reset),.clk  (clk));

ADDRESS_FSM

address_FSM (.a_data (a_data),.dp_l_read (dp_l_read),
.address (address),.reset (reset),.clk  (clk));

always @ (posedge clk) begin: Out_Mux:
case (sel)
     0 : s_data = d_data; // select descriptor data.
     1 : s_data = c_data; // select config data.
     2 : s_data = a_data; // select address data
     3 : s_data = other; // select other appropriate data.
endcase
end
wire [ 7:0 ] dpdi = (usb_endpt == 0) ? AD_out : s_data;
endmodule
```

FIG. 6A

```
module SETUP_FSM (dpdout, clk, reset, data, mux,
dp_l_write, set_token, valid_token);

input [7:0] dpdout;
input [3:0] usb_endpt;
input clk, reset, valid_token, set_token, dp_l_write;

output [7:0] data;
output interpret; reg interpret;

reg[7:0] REQUEST, REQ, VALUE1, VALUE2, INDEX1,
         INDEX2, LENGTH1, LENGTH2;
reg[3:0] state, next_state;

always @ (negedge clk)
  if (set_token & valid_token)  state = get_SETUP;
  else state = next_state;
always @ (posedge dp_l_write or posedge reset)
begin: SETUP_FSM
  if (reset) begin
      next_state = get_SETUP;
      interrupt <= 0;
      end
else if (valid control endpoint and set_token)
begin case (state)
get_SETUP : begin
            interrupt <= 0;
            REQ[7]   = dpdout[7];
            REQ[6:5] = {dpdout[6], dpdout[5]};
            next_state = get_REQ;
            end
get_REQ : begin
            REQUEST = dpdout;  // [7:0]
            next_state = get_VALUE1
            end
get_VALUE1: begin
            VALUE1 = dpdout;
            next_state = get_VALUE2;
            end
get_VALUE2: begin
            VALUE2 = dpdout;
            next_state = get_INDEX1;
            end
get_INDEX1:
get_INDEX2:
get_LENGTH1: begin
               LENGTH1 = dpdout;
               next_state = get_LENGTH2;
                        end
get_LENGTH2: begin
               LENGTH2 = dpdout;
               next_state = get_LENGTH2;
               interpret <= 1; // enable interpret_FSM
               end
endcase

endmodule
```

FIG. 6B

```
module INTERPRET_FSM (clk, reset, data,
stall, mux, sel, address, config,
descrip, DIR, STD, etc.);
input [7:0] data;
input interpret;
output [2:0] mux, sel;
output address, config, descrip; // or
other interpreted requests
output DIR, STD, etc;  // CLASS, VENDOR,
etc.

reg address, config, descrip;
reg  [2:0]state, next_state;

always @ (state or interpret or data)
begin : INTERPRET
if  (interpret) begin

case (state)

INIT: begin
      stall = 0;
      mux = REQ_type:
      next_state = got_REQ_type;
      end

got_REQ_type: begin

      DIR <= data[7];

case ({ data[6], data[5] })

      00 : STD = 1;
      01 : CLAS = 1;
      10 : VEND = 1;
      11 : ERR = 1;

endcase
      next_state = get_REQ;
end

get_REQ:    begin
            mux = sel_REQ;
            next_state = got_REQ;
            end
got_REQ:    begin

case (data)

      00: begin
            sel = `STATUS;
            status = 1;
            next_state = get_status;
        end

      05: begin
            sel = `ADDRESS;
            address = 1;
            next_state = get_address
            end

      06: begin
            sel = `DESCRIP;
            descrip = 1;
            next_state = get_desc
            end

      08:  begin
             sel = `CONFIG;
             do_config = 1;
             next_state = get_config;
             end

      0A: begin
             sel = `INTERFACE;
             interface = 1;
             next_state = get_interface;
             end

default:  stall = 1;

endcase //data
end

get_address : begin
mux = VALUE2;
if (interpret) next_state = get_address;
else  next_state = INIT:
end

endcase //state

end


always @ (posedge clk) begin
  if(reset) state = INIT;
  else state = next_state;
                        end

endmodule
```

FIG 6C

```
Module  DESCRIP_FSM (desc, descrip, dp_l_read)

output [7:0] desc;
input              descrip, dp_l_read,

always @ (posedge dp_l_read) begin: descrip_FSM

if (descrip) begin

case (state)

      0: begin
         desc = length; //18
         state = 1;
         end

      1: begin
          desc = type; // = 1
          state = 2;
          end

      2: begin
          desc = USB. version; //1
          state = 3;
          end

      3: begin
          desc = next_byte, etc.
          state = 4;
          end

      states 4 thru 16 not shown

      17: begin
            desc = last byte
            state = last_state;
            end

     last state: begin . . .

endcase

end

endmodule
```

FIG. 6D

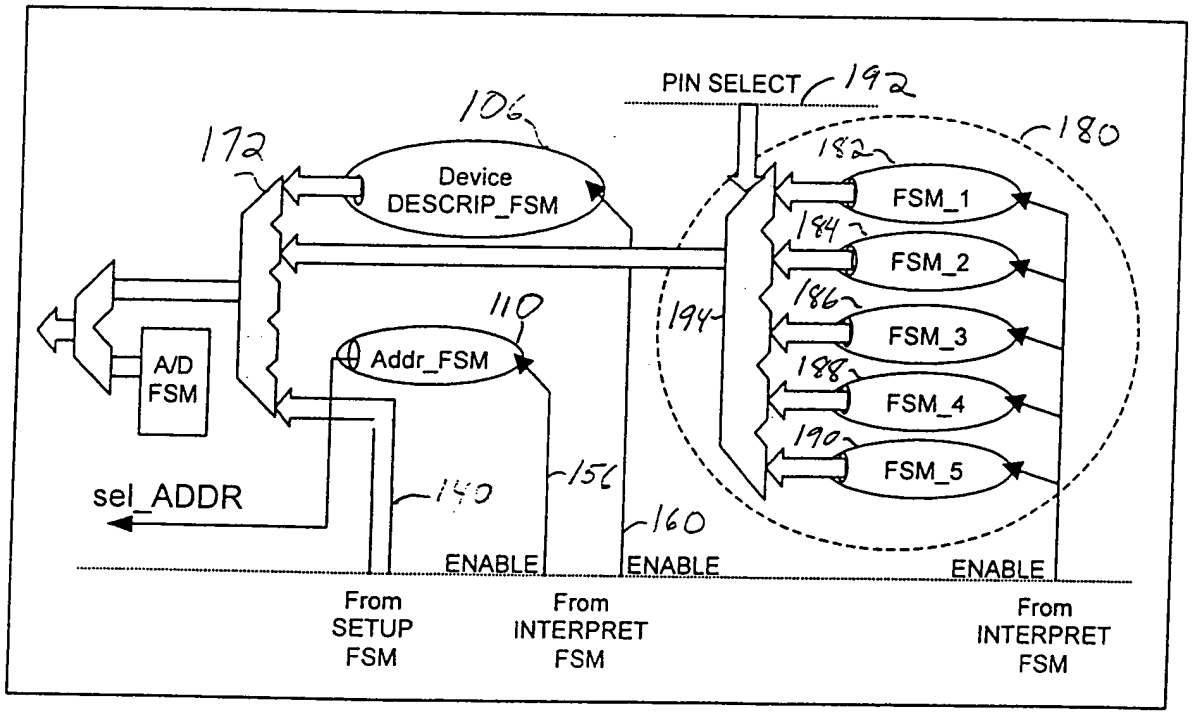
PIN SELECT
192
106
172
180
182
Device
DESCRIP_FSM
FSM_1
184
FSM_2
186
FSM_3
188
FSM_4
190
FSM_5
194
110
A/D
FSM
Addr_FSM
sel_ADDR
140
156
160
ENABLE
ENABLE
ENABLE
From
SETUP
FSM
From
INTERPRET
FSM
From
INTERPRET
FSM

FIG. 7

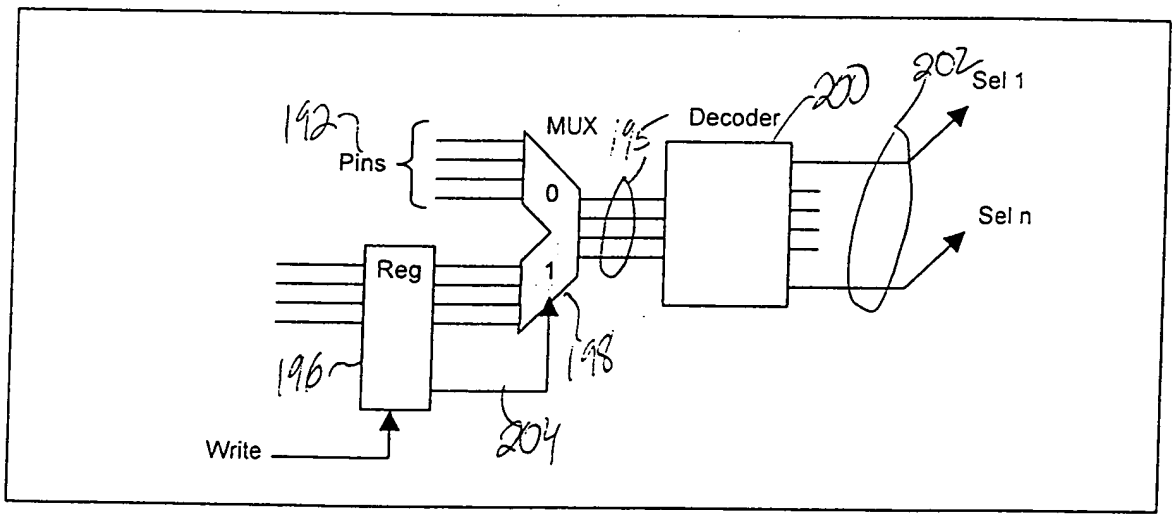
192
Pins
MUX
195
Decoder
200
202
Sel 1
Sel n
0
1
Reg
196
198
204
Write

FIG. 8

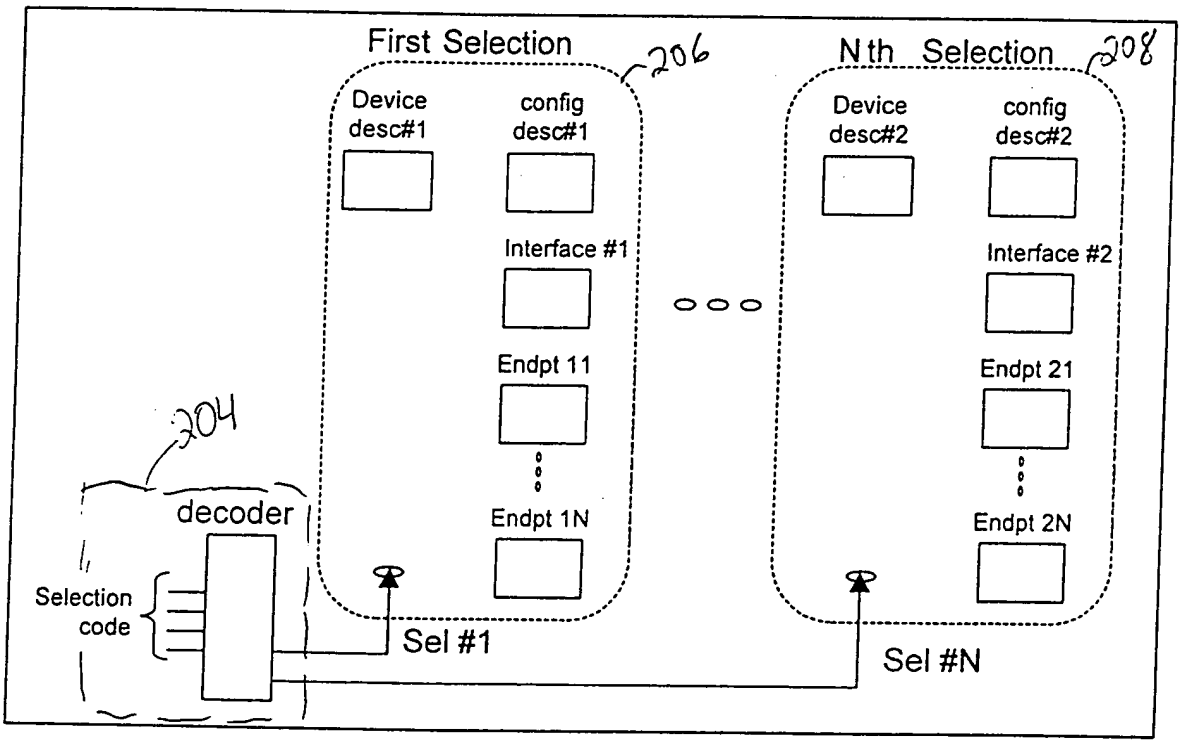
First Selection
206
Nth Selection
208
Device desc#1
config desc#1
Interface #1
Endpt 11
Endpt 1N
Device desc#2
config desc#2
Interface #2
Endpt 21
Endpt 2N
204
decoder
Selection code
Sel #1
Sel #N

FIG. 9

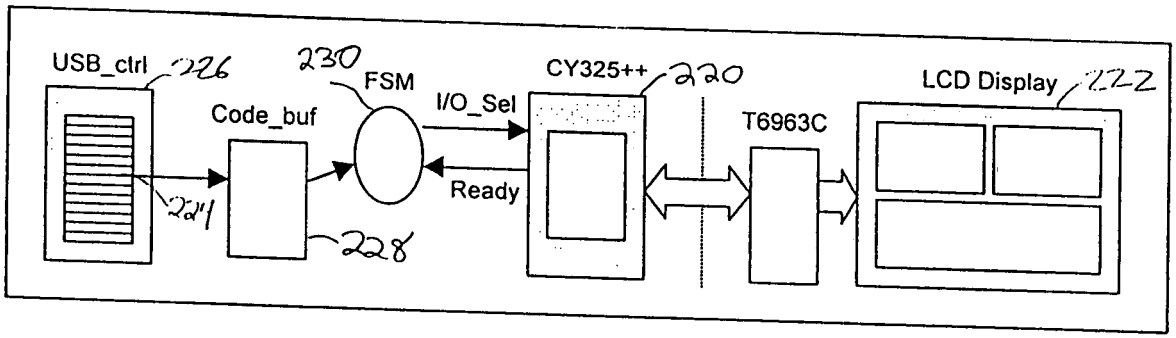
USB_ctrl
226
230
FSM
Code_buf
I/O_Sel
Ready
CY325++
220
T6963C
LCD Display
222
224
228

FIG 10

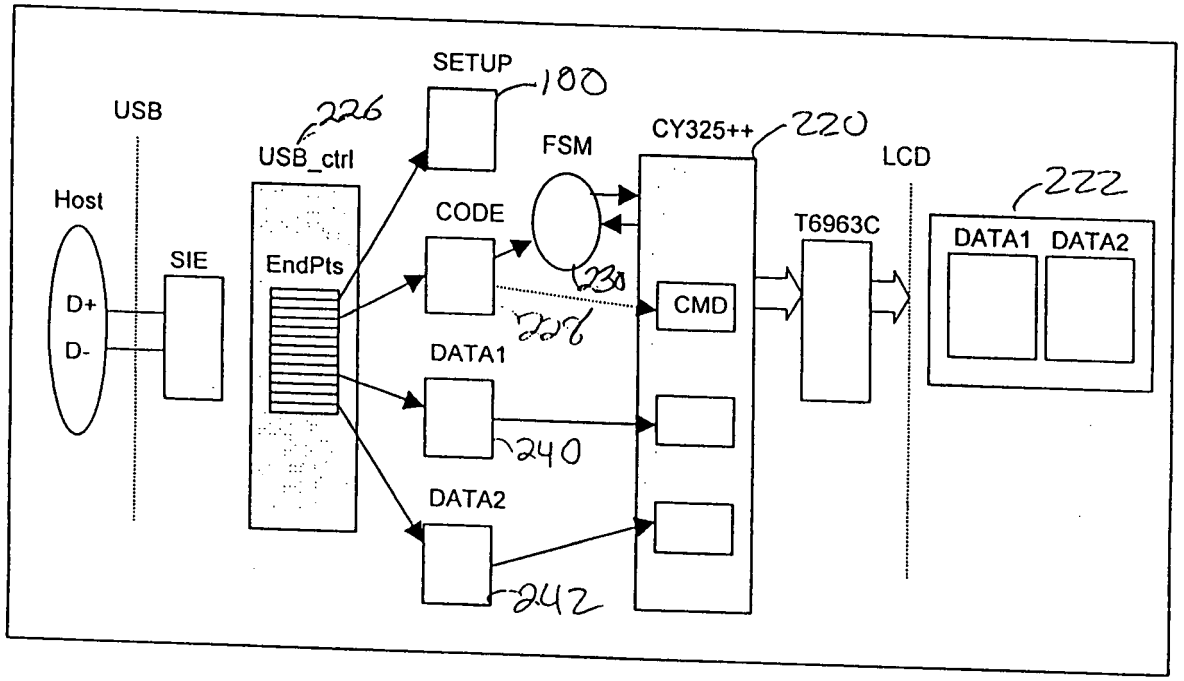

FIG. 11